

1. [15] This question concerns a function `containsOdd :: [Int] -> Bool`, which is defined so that `containsOdd ns` is `True` if and only if any of the elements of `ns` is odd.

a. Fill in the blank to define `containsOdd` using `map`.

```
> containsOdd ns = or (map odd ns)
```

b. Fill in the blank to define `containsOdd` using `filter` and function composition.

```
> containsOdd = not . null . filter odd
```

c. Fill in the blank to define `containsOdd` using `foldr`.

```
> containsOdd = foldr (\x bs -> odd x || bs) False
```

2. [10] Define a Haskell program `cat` which (like the unix utility `cat`) copies characters, one line at a time, from standard input to standard output. To detect end of input, use `isEOF :: IO Bool` (defined in the library module `IO`), which delivers `True` if standard input is empty and `False` otherwise.

```
> import IO
> cat :: IO ()
> cat = do end <- isEOF
>         if end
>           then return ()
>           else do line <- getLine
>                   putStrLn line
>                   cat
```

3. [15] Given the following grammar:

```
Alpha ::= Alpha @ Beta | Beta
Beta    ::= Gamma | Gamma % Beta
Gamma   ::= Let { Let } | < Alpha >
Let     ::= a | b | ... | z
```

use definitions in the library `ParseLib.hs` to define a Haskell parser which translates expressions that obey the grammar into postfix expressions [in the form of `Strings`]. Postfix expressions' subexpressions should be separated by space characters.

```
> import ParseLib

> alpha :: Parser String
> alpha = do y <- beta
>           more y
>           where
>             more x = do char '@'
>                         y <- beta
>                         more (x ++ " " ++ y ++ "@")
>                         +++
>                         return x

> beta :: Parser String
> beta = do s <- gamma
>           char '%'
>           y <- beta
>           return (s ++ " " ++ y ++ "%")
>           +++
>           gamma

> gamma :: Parser String
> gamma = do char '<'
>            k <- alpha
>            char '>'
>            return k
>           +++
>           do c <- lower
>              cs <- many lower
>              return (c:cs)
```

4. [15] Given these definitions

$$\begin{aligned} \text{init } [x] &= [] \\ \text{init } (x:y:ys) &= x : \text{init } (y:ys) \end{aligned}$$
$$\begin{aligned} \text{last } [x] &= x \\ \text{last } (x:y:ys) &= \text{last } (y:ys) \end{aligned}$$
$$\begin{aligned} [] ++ ys &= ys \\ (x:xs) ++ ys &= x : (xs ++ ys) \end{aligned}$$

consider the proposition

$$\text{init } as ++ [\text{last } as] = as$$

a. For what lists as is this proposition true?

For all non-empty lists.

b. Prove that it holds for those lists.

Case $[a]$:

$$\begin{aligned} &\text{init } [a] ++ [\text{last } [a]] = [a] \\ = &\quad \{ \text{defs. of init and last} \} \\ &[] ++ [a] = [a] \\ = &\quad \{ \text{def. of } (++) \} \\ &[a] = [a] \\ = &\quad \{ \text{identity} \} \\ &\text{true} \end{aligned}$$

Case $(a:b:bs)$

$$\begin{aligned} &\text{init } (a:b:bs) ++ [\text{last } (a:b:bs)] \\ = &\quad \{ \text{defs. of init and last} \} \\ &(a : \text{init } (b:bs)) ++ [\text{last } (b:bs)] \\ = &\quad \{ \text{def. of } (++) \} \\ &a : (\text{init } (b:bs) ++ [\text{last } (b:bs)]) \\ = &\quad \{ \text{induction hypothesis} \} \\ &a : (b:bs) \\ = &\quad \{ (:) \text{ is right-associative} \} \\ &a:b:bs \quad \blacksquare \end{aligned}$$

5. [15] Find the weakest precondition such that execution of

$$a := b+2; a, b, c := b+c, c+a, a+b$$

establishes $a \leq b < c$.

$$\begin{aligned} & \text{wp}("a := b+2; a, b, c := b+c, c+a, a+b", a \leq b < c) \\ = & \quad \{ \text{def. of ;} \} \\ & \text{wp}("a := b+2", \text{wp}("a, b, c := b+c, c+a, a+b", a \leq b < c)) \\ = & \quad \{ \text{def. of :=} \} \\ & \left((a \leq b < c)_{b+c, c+a, a+b}^{a, b, c} \right)_{b+2}^a \\ = & \quad \{ \text{substitution} \} \\ & (b+c \leq c+a < a+b)_{b+2}^a \\ = & \quad \{ \text{substitution} \} \\ & b+c \leq c+b+2 < b+2+b \\ = & \quad \{ \text{conjunctuality} \} \\ & b+c \leq c+b+2 \wedge c+b+2 < b+2+b \\ = & \quad \{ \text{simplification} \} \\ & b \leq b+2 \wedge c < b \\ = & \quad \{ \text{simplification} \} \\ & c < b \end{aligned}$$

6. [15] Define GCN commands equivalent to each of the following C statements:

a. `if(x > y) z = x+1;`

if $x > y \rightarrow z := x+1 \quad \parallel \quad x \leq y \rightarrow \text{skip}$ **fi**

b. `while(n > j)
 for(j = 0, k = 1; k < m; j=j+1, k=k+2);
 n--;`

do $n > j \rightarrow j, k := 0, 1;$
 do $k < m \rightarrow j, k := j+1, k+2$ **od**
od;
 $n := n-1$

c. `switch(x) {
 case 1: x = x + a;
 case 2: x = x + b;
 case 3: x = x + c;
}`

if $x = 1 \rightarrow x := x + a + b + c$
 \parallel $x = 2 \rightarrow x := x + b + c$
 \parallel $x = 3 \rightarrow x := x + c$
 \parallel $x \neq 1 \wedge x \neq 2 \wedge x \neq 3 \rightarrow \text{skip}$
fi